
Безопасность ядра Linux:
в теории и на практике
Александр Попов
Positive Technologies
HL
HighLoad++
2022

Безопасность ядра Linux: в теории и на практике

Александр Попов

Positive Technologies

25.11.2022



Кто я

- Александр Попов
- Разработчик ядра Linux с 2012 года
- Исследователь информационной безопасности в

  positive technologies
- Докладчик на конференциях:

  OffensiveCon, Nullcon, Linux Security Summit, Still Hacking Anyway, Zer0Con

  Positive Hack Days, ZeroNights, Open Source Summit, OS DAY, Linux Plumbers

  и других

  a13xp0p0v.github.io/conference_talks

Цель доклада

1. Показать вам общую картину безопасности ядра Linux
2. Рассказать об инструментах, которые помогают:
   - изучать эту предметную область
   - управлять безопасностью ядра Linux



Определимся с терминами

Операционная система (ОС) — это программное обеспечение, которое:

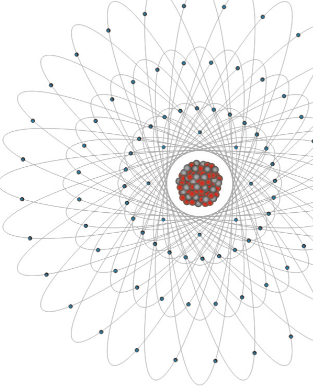
1. управляет аппаратными и программными ресурсами компьютера
2. предоставляет сервисы для компьютерных программ

Определимся с терминами

Ядро ОС — основная (и самая интересная) часть ОС, управляющая процессами и их доступом к ресурсам вычислительной системы:

- процессорному времени (задача планировщика)
- оперативной памяти
- аппаратным средствам
- механизмам межпроцессного взаимодействия (IPC)

Ядро работает в привилегированном режиме CPU

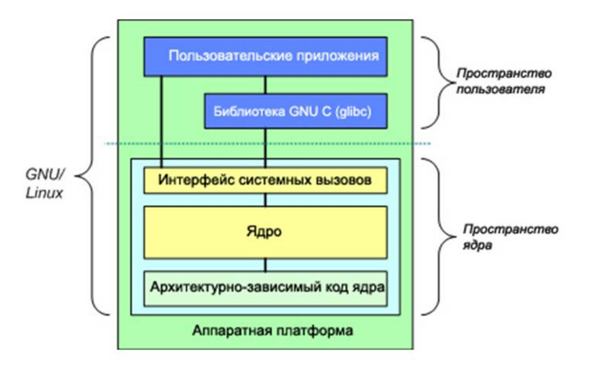


pediaa.com/difference-between-uranium-and-thorium

Linux Kernel

- **Ядро Linux** — основа семейства операционных систем GNU/Linux и Android
- «One of the most successful collaborative development projects in history»
  linuxfoundation.org/2017-linux-kernel-report-landing-page
  - 90% публичных облачных ресурсов (2017)
  - 62% рынка встраиваемых устройств (2017)
  - 99% рынка суперкомпьютеров (2017)
  - Более 3 млрд активных Android-устройств (2021)
    theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021
  - Разрабатывается под открытой лицензией GPL 2.0
  - Более 4000 разработчиков принимают участие ежегодно с 2017 года

Интерфейсы ядра Linux



ibm.com/developerworks/linux/library/l-linux-kernel/

Определимся с терминами

- Основная задача информационной безопасности — целесообразная и сбалансированная защита конфиденциальности, целостности и доступности данных
- «Безопасность — это управление рисками» (Брюс Шнайер)
  schneier.com/essays/archives/2007/01/information_security_1.html
- Для оценки рисков необходима модель угроз информационной системы
- Без модели угроз невозможно выработать хорошую модель безопасности (как средства защиты устраняют угрозы и снижают риски)

Определимся с терминами

- **Уязвимость** — недостаток в программно-аппаратном обеспечении информационной системы, который может быть использован (проэксплуатирован) для реализации угрозы безопасности (проведения атаки)
- **Эксплойт** — программа или последовательность команд, использующая уязвимости в ПО и применяемая для атаки на информационную систему

Устранение уязвимостей в ядре Linux

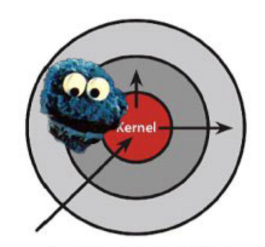
- Гигантская кодовая база ядра Linux развивается с огромной скоростью.
  Linux v5.11:
  - Содержит более 30 млн строк кода
  - 8900 строк кода добавляется, 2500 удаляется и 2100 изменяется **каждый день**
  - Средняя скорость merge — 9,6 патчей в час
  - Каждый год в развитии участвуют более 4000 разработчиков (с 2017 года)

- У нас есть санитайзеры, фаззер syzkaller, инструменты статического анализа, но...

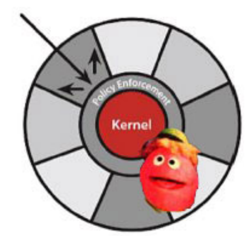
- Уязвимости появляются быстрее, чем исправляются
  («Сказка о тысяче ядерных багов» Дмитрия Вьюкова)

Уязвимости и механизмы разграничения доступа

- Механизмы разграничения доступа в ядре Linux реализуются с помощью LSM (Linux Security Modules)
- Примеры LSM из «ванильного» ядра: AppArmor, SELinux, Smack
- LSM не защищают систему от эксплуатации ядерных уязвимостей
- Ниже — объяснение от grsecurity



Discretionary Access Control
Once a security exploit gains access to priveleged system component, the entire system is compromised.



Mandatory Access Control
Kernel policy defines application rights, firewalling applications from compromising the entire system.



Blackhats with kernel exploits
Basement dwelling 12-year olds armed with kernel exploit released past Tuesday. A SELinux disabling payload in the exploit turns your entire MAC policy into laughing stock. You spend the rest of the weekend removing SSH backdoors.

Kernel Self Protection Project

- Чтобы повысить безопасность ядра, нужно больше чем исправление ошибок
- Ядро Linux должно безопасно отрабатывать в ошибочной ситуации
- Идеи **grsecurity и PaX** — во многом источник вдохновения
- **Цель**: устранение классов уязвимостей и методов их эксплуатации
  - KSPP wiki: kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
  - Обзор KSPP (Kees Cook): outflux.net/slides/2021/lss/kspp.pdf



Безопасность ядра Linux

Безопасность ядра Linux — очень сложная предметная область. Ключевые понятия:

- Классы уязвимостей
- Техники эксплуатации уязвимостей
- Механизмы выявления ошибок
- Технологии защиты
  - Входящие в mainline
  - Поставляемые отдельно (в т. ч. коммерческие)
  - Требующие аппаратной поддержки



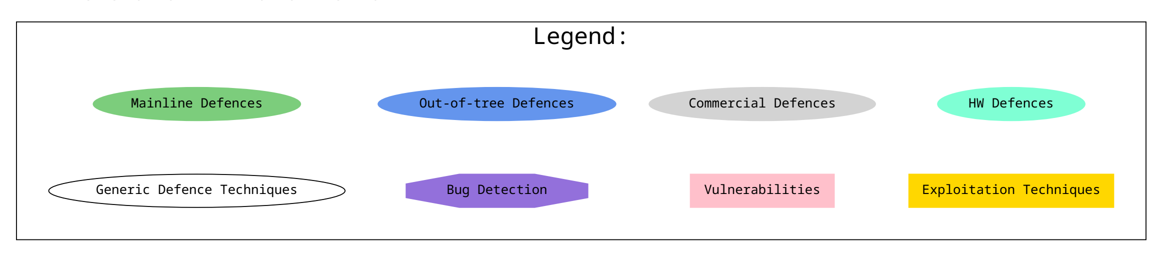
Drawn by Daniel Reeve, made by weta

Все они имеют сложные взаимосвязи...

Было бы полезно иметь их графическое представление

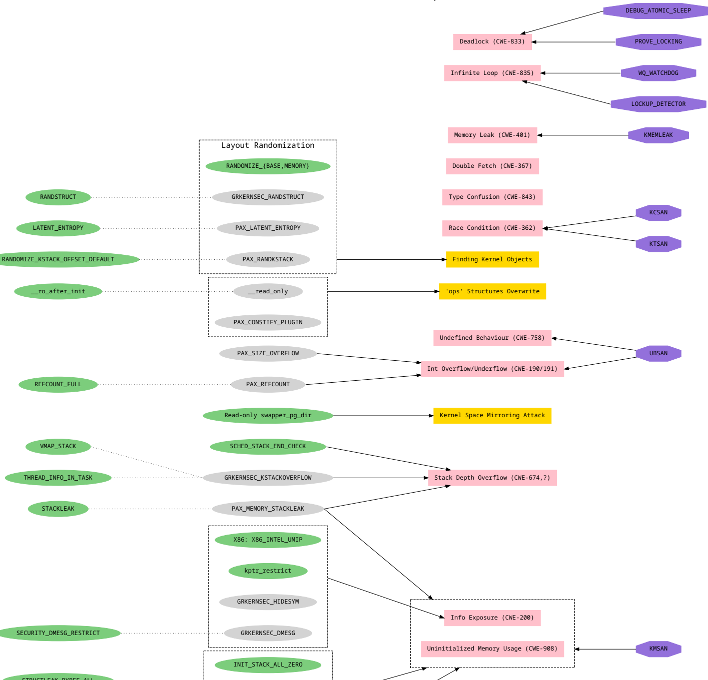
Linux Kernel Defence Map

- Поэтому я разработал **карту средств защиты ядра Linux**
  github.com/a13xp0p0v/linux-kernel-defence-map
- Начал создавать карту в 2018 году, продолжаю улучшать и обновлять ее
- На схеме — ключевые понятия:



- Каждая линия между объектами на карте обозначает их взаимное влияние
- Суть этого влияния следует выяснять в документации
- [!] Карта не затрагивает способы уменьшения поверхности атаки

Linux Kernel Defence Map, whole picture (1/5)



Linux Kernel Defence Map, whole picture (2/5)

Linux Kernel Defence Map, whole picture (3/5)

PAX_KERNEXEC
XPFO
ZERO_CALL_USED_REGS
SMEP/PXN
ARM: CFI_CLANG
PAX_RAP
ARM: ARM64_PTR_AUTH_KERNEL
ARM: SHADOW_CALL_STACK
ARM: ARM64_BTI_KERNEL
X86: X86_SHADOW_STACK
X86: Intel CET
X86: X86_KERNEL_IBT
ARM: ARM64_SW_TTBR0_PAN
ARM: CPU_SW_DOMAIN_PAN

Control Flow Integrity
Fine-grained Forward-edge CFI
Backward-edge CFI
Coarse-grained Forward-edge CFI

Control Flow Hijack Techniques
ret2dir
ret2dir + ROP/JOP/COP
ROP/JOP/COP
ret2usr + ROP/JOP/COP
ret2usr
Return Address Overwrite

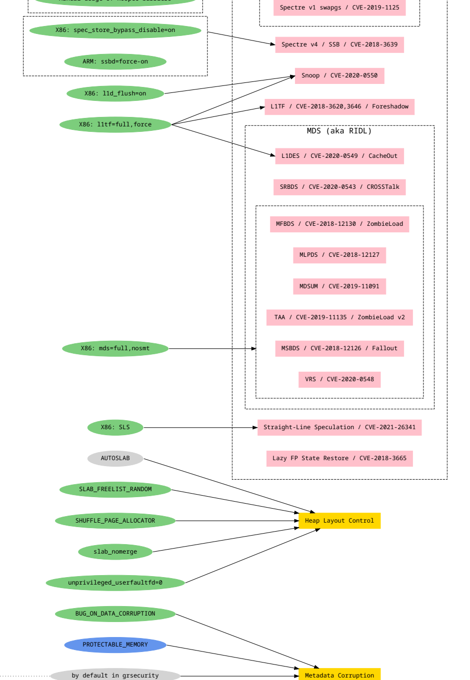
STACKPROTECTOR
SMAP/PAN
PAX_UDEREF
DEFAULT_MMAP_MIN_ADDR=65536

Userspace Data Access
NULL Pointer Dereference (CWE-476)

X86: pti=on (PAGE_TABLE_ISOLATION)
ARM: kpti=on (UNMAP_KERNEL_AT_EL0)
ARM: HARDEN_BRANCH_PREDICTOR
X86: spectre_v2=on (RETPOLINE)
ARM: MITIGATE_SPECTRE_BRANCH_HISTORY
X86: MICROCODE
mitigations=auto,nosmt
X86: SCHED_CORE
RESPECTRE
Manual usage of nospec barriers

Transient Execution Vulnerabilities (Covert Channels, CWE-514)
Spectre v3 / RDCL / CVE-2017-5754 / Meltdown
Spectre v2 / BTI / CVE-2017-5715
Spectre RSB / CVE-2018-15572
BHI, IMBTI / CVE-2022-0001,0002
Spectre-BHB / CVE-2022-23960
Spectre v3a / RSRE / CVE-2018-3640
Spectre v1 / BCB / CVE-2017-5753
Spectre v1.1 / BCBS / CVE-2018-3693
Spectre v1 swapgs / CVE-2019-1125

Linux Kernel Defence Map, whole picture (4/5)



Linux Kernel Defence Map, whole picture (5/5)

MFBDS / CVE-2018-12130 / ZombieLoad

MLPDS / CVE-2018-12127

MDSUM / CVE-2019-11091

TAA / CVE-2019-11135 / ZombieLoad v2

X86: mds=full,nosmt

MSBDS / CVE-2018-12126 / Fallout

VRS / CVE-2020-0548

X86: SLS

Straight-Line Speculation / CVE-2021-26341

AUTOSLAB

Lazy FP State Restore / CVE-2018-3665

SLAB_FREELIST_RANDOM

SHUFFLE_PAGE_ALLOCATOR

Heap Layout Control

slab_nomerge

unprivileged_userfaultfd=0

BUG_ON_DATA_CORRUPTION

PROTECTABLE_MEMORY

STATIC_USERMODEHELPER

by default in grsecurity

Metadata Corruption

DEBUG_{LIST,SG,CREDENTIALS,NOTIFIERS,VIRTUAL}

Linux Kernel Runtime Guard

Changing Kernel Image

LOCKDOWN_LSM

Legend:

Mainline Defences

Out-of-tree Defences

Commercial Defences

HW Defences

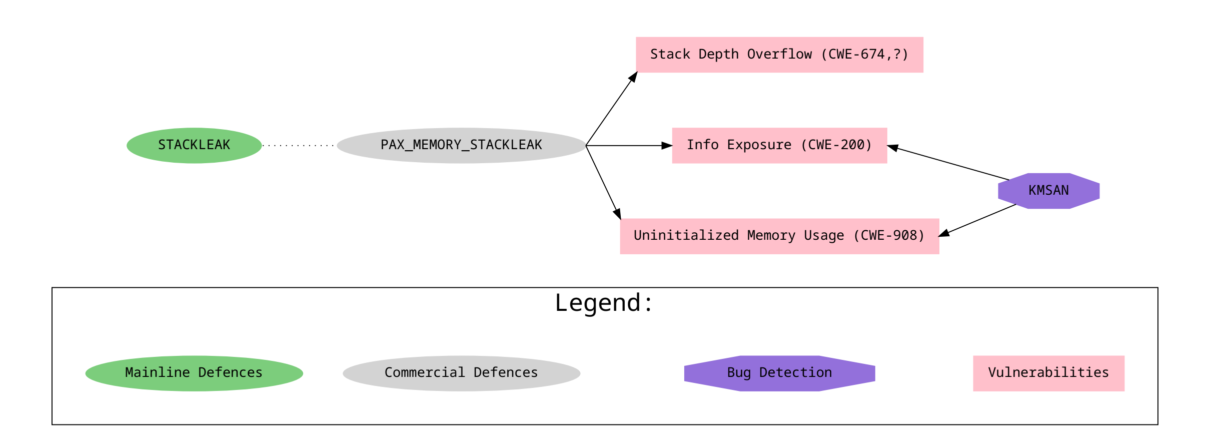
Generic Defence Techniques

Bug Detection

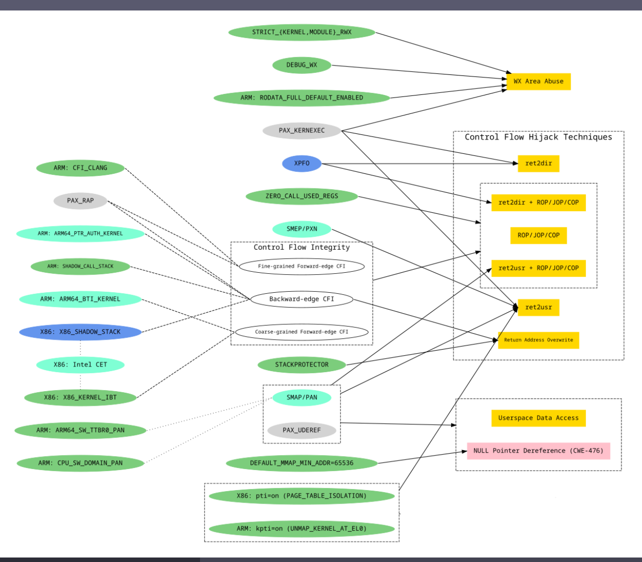
Vulnerabilities

Exploitation Techniques

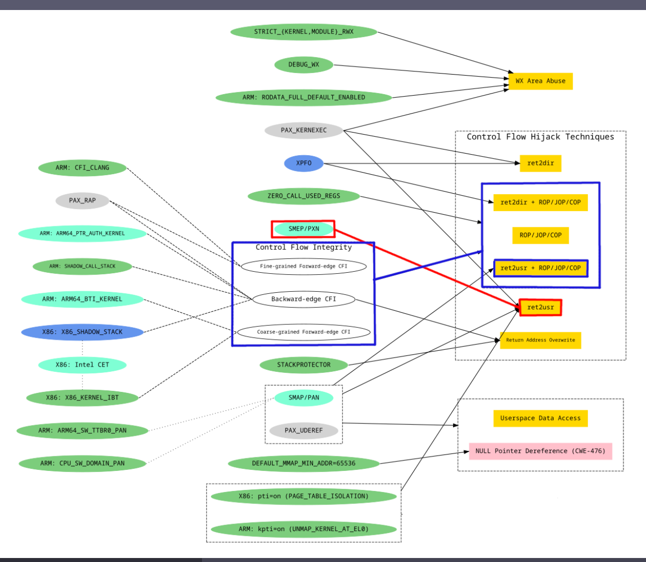
Примеры из карты: STACKLEAK



Примеры из карты: Control-Flow Hijack — перехват потока управления

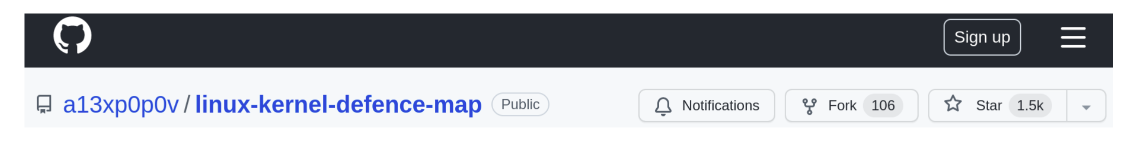


Примеры из карты: Control-Flow Hijack — перехват потока управления



Linux Kernel Defence Map: реализация

- Карту нужно обновлять (ядро Linux развивается)
- Желательно иметь исходник в текстовом виде и вести его в VCS
- Не хочется вручную расставлять объекты (с минимальным количеством пересечений связей)
- Поэтому я пишу карту на языке DOT, схему генерирую с помощью Graphviz:
  # dot -Tsvg map.dot -o map.svg
- Проект живой и успешный, участие приветствуется



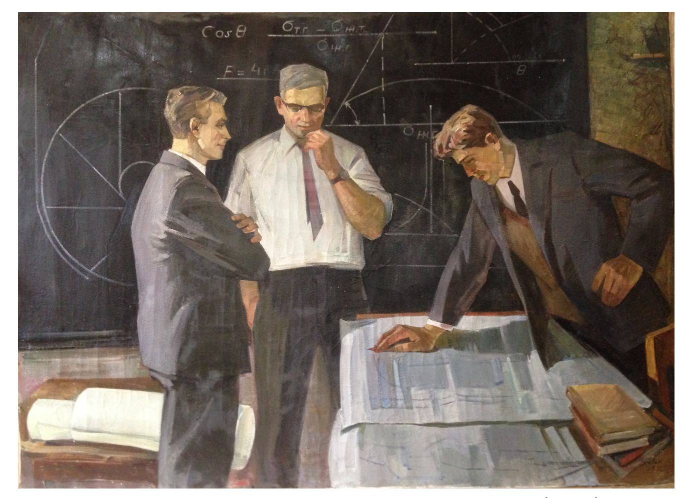
Пример кода карты

```
// Defences relations
edge [style=dotted, arrowhead=none, dir=none, headport=_, tailport=_];
"STACKLEAK":e -> "PAX_MEMORY_STACKLEAK":w;
// Bug Detection Mechanisms vs. Vulnerabilities
edge [style=solid, arrowhead=normal, dir=back, headport=_, tailport=_];
"Uninitialized Memory Usage (CWE-908)":e -> "KMSAN";
"Info Exposure (CWE-200)":e -> "KMSAN";
// Defences vs. Vulnerabilities and Exploitation Techniques
edge [style=solid, arrowhead=normal, dir=forward, headport=_, tailport=_];
"PAX_MEMORY_STACKLEAK":e -> "Stack Depth Overflow (CWE-674,?)":sw;
"PAX_MEMORY_STACKLEAK":e -> "Uninitialized Memory Usage (CWE-908)":nw;
"PAX_MEMORY_STACKLEAK":e -> "Info Exposure (CWE-200)":w;
```

Источники для Linux Kernel Defence Map

- Документация ядра Linux (раздел о безопасности)
- Документация grsecurity
- Рекомендации Kernel Self Protection Project
- Публикации Microsoft Security Response Center (MSRC)
- Другие источники: github.com/a13xp0p0v/linux-kernel-defence-map#references

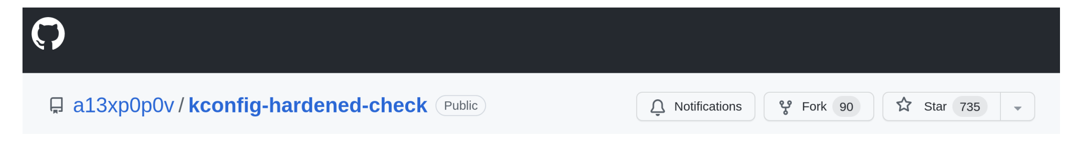
Классная карта! А что на практике?



В. Е. Белов. «Советские ученые-теоретики» (1972)

Параметры безопасности ядра

- Есть **огромное количество** параметров безопасности ядра Linux
- В популярных дистрибутивах многие из этих параметров **не настроены**
- Проверять конфигурации вручную **любят не все** (или **не любят все**?)
- **Так пусть компьютеры делают свою работу!**
- Я создал проект **kconfig-hardened-check** для проверки параметров безопасности ядра Linux: github.com/a13xp0p0v/kconfig-hardened-check
- Работа началась в 2018 году, идет активная разработка



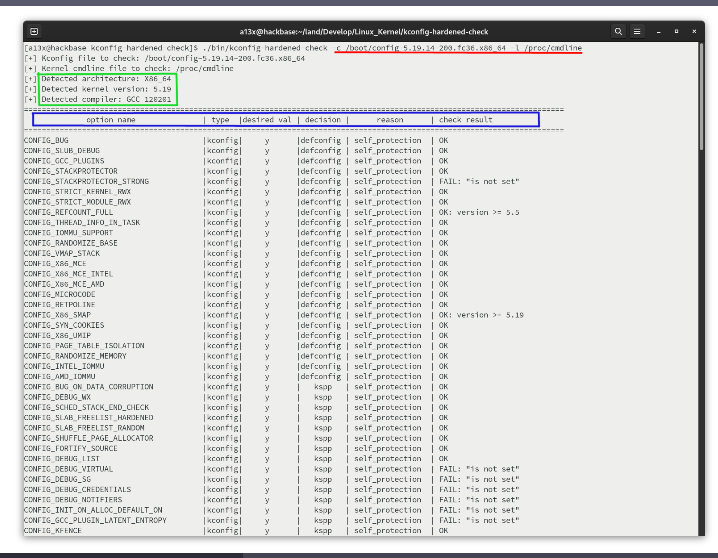
kconfig-hardened-check -h

```
[a13x@hackbase kconfig-hardened-check]$ ./bin/kconfig-hardened-check
usage: kconfig-hardened-check [-h] [--version] [-p {X86_64,X86_32,ARM64,ARM}] [-c CONFIG]
[-l CMDLINE] [-m {verbose,json,show_ok,show_fail}]

A tool for checking the security hardening options of the Linux kernel

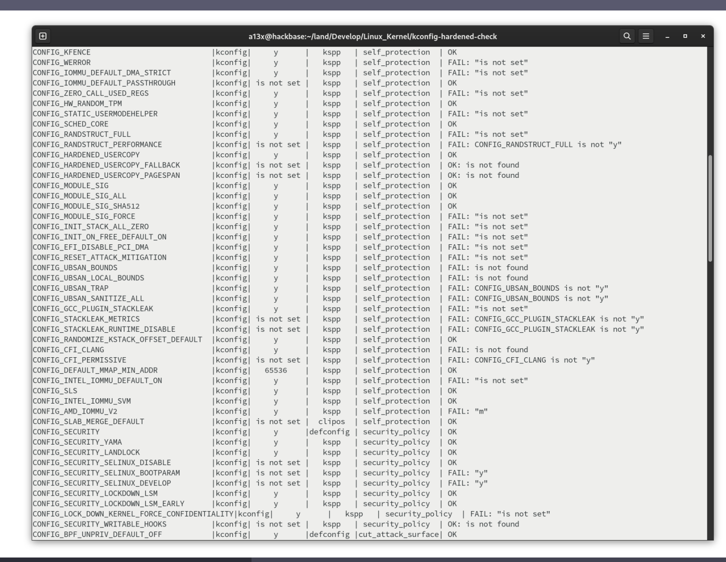
optional arguments:
  -h, --help            show this help message and exit
  --version       1     show program's version number and exit
  -p {X86_64,X86_32,ARM64,ARM}, --print {X86_64,X86_32,ARM64,ARM}
         2              print security hardening preferences for the selected architecture
  -c CONFIG, --config CONFIG
          3             check the kernel kconfig file against these preferences
  -l CMDLINE, --cmdline CMDLINE
                4       check the kernel cmdline file against these preferences
  -m {verbose,json,show_ok,show_fail}, --mode {verbose,json,show_ok,show_fail}
                        choose the report mode
```

kconfig-hardened-check: пример вывода (1/5)

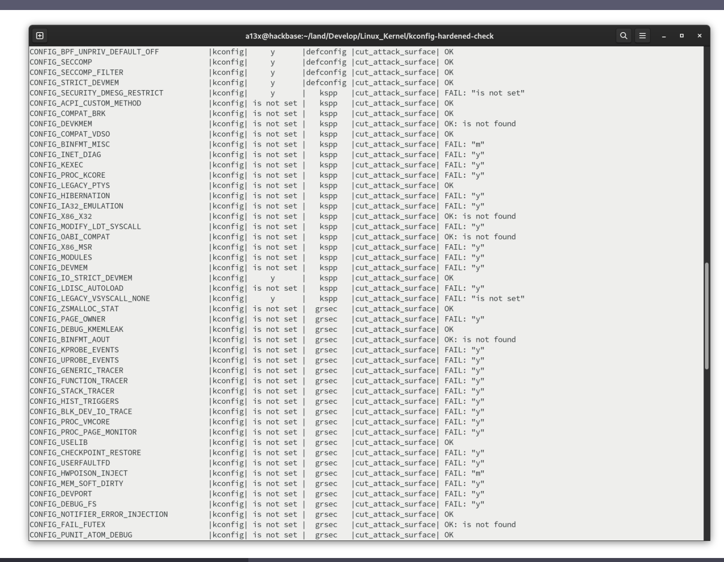


```
a13x@hackbase:~/land/Develop/Linux_Kernel/kconfig-hardened-check
[a13x@hackbase kconfig-hardened-check]$ ./bin/kconfig-hardened-check -c /boot/config-5.19.14-200.fc36.x86_64 -l /proc/cmdline
[+] Kconfig file to check: /boot/config-5.19.14-200.fc36.x86_64
[+] Kernel cmdline file to check: /proc/cmdline
[+] Detected architecture: X86_64
[+] Detected kernel version: 5.19
[+] Detected compiler: GCC 120201
              option name               | type  |desired val | decision |      reason      | check result
CONFIG_BUG                              |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_SLUB_DEBUG                       |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_GCC_PLUGINS                      |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_STACKPROTECTOR                   |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_STACKPROTECTOR_STRONG            |kconfig|     y      |defconfig | self_protection  | FAIL: "is not set"
CONFIG_STRICT_KERNEL_RWX                |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_STRICT_MODULE_RWX                |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_REFCOUNT_FULL                    |kconfig|     y      |defconfig | self_protection  | OK: version >= 5.5
CONFIG_THREAD_INFO_IN_TASK              |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_IOMMU_SUPPORT                    |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_RANDOMIZE_BASE                   |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_VMAP_STACK                       |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_X86_MCE                          |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_X86_MCE_INTEL                    |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_X86_MCE_AMD                      |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_MICROCODE                        |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_RETPOLINE                        |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_X86_SMAP                         |kconfig|     y      |defconfig | self_protection  | OK: version >= 5.19
CONFIG_SYN_COOKIES                      |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_X86_UMIP                         |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_PAGE_TABLE_ISOLATION             |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_RANDOMIZE_MEMORY                 |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_INTEL_IOMMU                      |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_AMD_IOMMU                        |kconfig|     y      |defconfig | self_protection  | OK
CONFIG_BUG_ON_DATA_CORRUPTION           |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_DEBUG_WX                         |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_SCHED_STACK_END_CHECK            |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_SLAB_FREELIST_HARDENED           |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_SLAB_FREELIST_RANDOM             |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_SHUFFLE_PAGE_ALLOCATOR           |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_FORTIFY_SOURCE                   |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_DEBUG_LIST                       |kconfig|     y      |   kspp   | self_protection  | OK
CONFIG_DEBUG_VIRTUAL                    |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_DEBUG_SG                         |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_DEBUG_CREDENTIALS                |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_DEBUG_NOTIFIERS                  |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_INIT_ON_ALLOC_DEFAULT_ON         |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_GCC_PLUGIN_LATENT_ENTROPY        |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_KFENCE                           |kconfig|     y      |   kspp   | self_protection  | OK
```

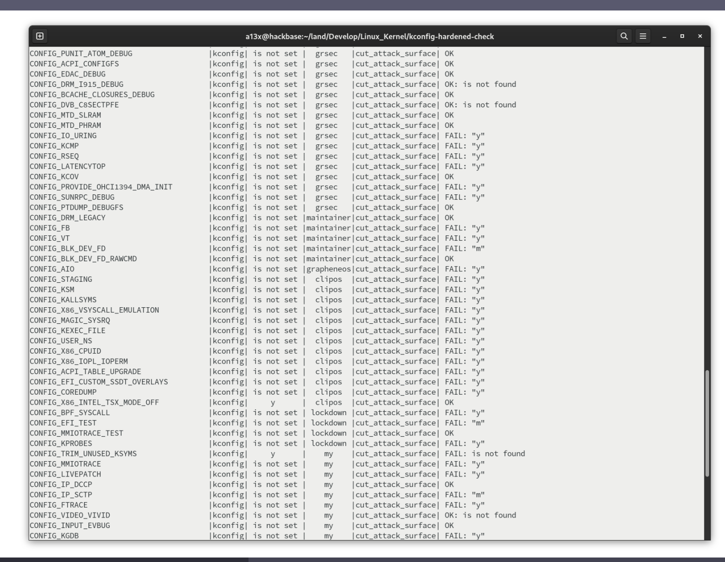
kconfig-hardened-check: пример вывода (2/5)



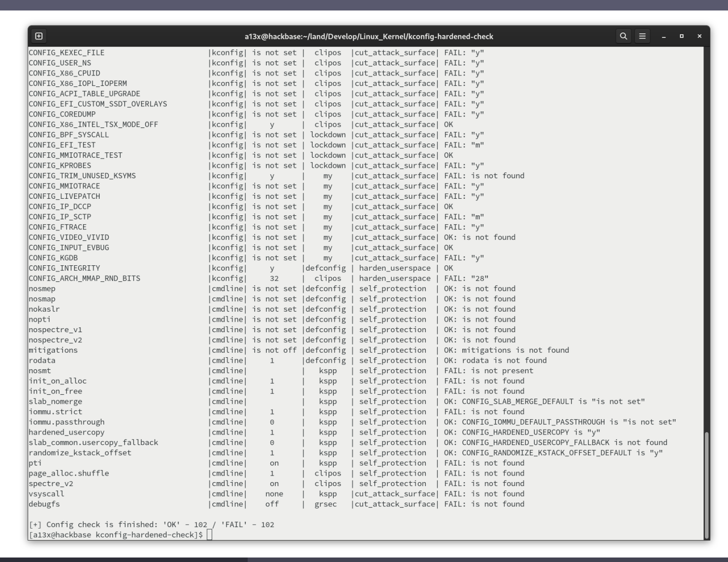
kconfig-hardened-check: пример вывода (3/5)



kconfig-hardened-check: пример вывода (4/5)



kconfig-hardened-check: пример вывода (5/5)

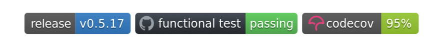


Источники для kconfig-hardened-check

- Рекомендации KSPP
- Конфигурация ядра CLIP OS
- Последний публичный патч grsecurity (отключаемые параметры)
- SECURITY_LOCKDOWN_LSM
- Обратная связь непосредственно от мейнтейнеров ядра Linux

kconfig-hardened-check: что под капотом

- Лицензия GPL 3.0
- Код на Python. Извините, мой код выглядит, как код на C:
  это потому что я разработчик ядра :)
- Установка с помощью pip/setuptools
- Регулярные релизы (с привязкой к релизам ядра)
- CI: автоматические функциональные тесты с подсчетом покрытия



- В работах участвуют контрибьюторы из международного сообщества (спасибо!)
- Инструмент используется несколькими дистрибутивами GNU/Linux
  при выпуске пакета ядра (что очень радует)
- Технологический центр исследования безопасности ядра Linux
  поддерживает мой проект (спасибо!)

kconfig-hardened-check: идеи и планы

- Внедрить проверку динамических параметров безопасности ядра Linux:
  - kernel cmdline (WIP)
  - sysctl
- Реализовать возможность переопределения и расширения набора проверок
- Разработать инструмент для автоматизированной работы с Kconfig
- Добавить поддержку RISC-V
- Измерить накладные расходы производительности для рекомендуемых опций (зависят от типа рабочей нагрузки)
- Разработать документацию по параметрам безопасности ядра Linux

В. Е. Тихоненко. «Высотники. На новой стройке» (1975)

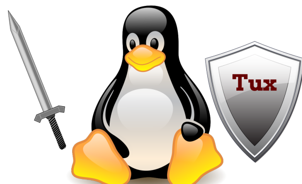


В заключение: как пользоваться этими инструментами

- «Безопасность — это управление рисками» (Брюс Шнайер)
- Для оценки рисков необходима модель угроз информационной системы
- Определив угрозы ИС на базе Linux,

  можно заняться разработкой ее модели безопасности:
  - С помощью Linux Kernel Defence Map определить релевантные средства защиты ядра
  - С помощью kconfig-hardened-check проверить и настроить соответствующие параметры
- Пример отличной модели безопасности: The Android Platform Security Model

  arxiv.org/abs/1904.05572



Спасибо! Вопросы?

Контакты:

alex.popov@linux.com

a13xp0p0v

positive technologies

POSIdev

Оценить доклад:

